2022 Rocky Mountain Regional Solutions

The Judges

Feb 25, 2023

Blueberry Waffle

Problem

- A waffle maker rotates 180 degrees every r seconds. A blueberry waffle is inserted with the blueberries pointing up. After f seconds, the waffle maker stops and rotates strictly fewer than 90 degrees back to horizontal. Are the blueberries pointing up or down?

Solution

- The waffle maker makes a full rotation every $2r$ seconds. Therefore, we can take f modulo $2r$. If f is less than $\frac{r}{2}$ or greater than $\frac{3r}{2}$, then the blueberries are pointing up. When f is equal to $\frac{r}{2}$ or $\frac{3r}{2}$, which is not allowed in the problem, the waffle maker is exactly vertical. When f is greater than $\frac{r}{2}$ and less than $\frac{3r}{2}$, the blueberries are pointing down.

Profitable Trip

Problem

- You are given a weighted, directed graph. You start at vertex 1 and travel some edges to get to vertex n. When you traverse an edge with weight t, you gain t dollars. However, you can only keep at most w more dollars than what you started out with - extra money is thrown away. What is the maximum profit you can make getting to vertex n?

Solution

- Because the weights here are small, you can explicitly maintain the maximum amount of money you can end with over all vertices, not just vertex n. The maximum loss you can incur is -2×10^5, so if you run Bellman-Ford over the graph, the maximum number of times you can relax a node is $2 \times 10^5 + 101$, which will run in time.
- Challenge: How do you solve this problem in the weights in the graph can go up to an absolute value of 10^9?

Champernowne Count

Problem

- The nth Champernowne word is obtained by concatenating the first n positive integers in order. Compute how many of the first n ($1 \leq n \leq 10^5$) Champernowne words are divisible by k ($1 \leq k \leq 10^9$).

Solution

- n is large enough that it is not practical to store the integers using arbitrary precision integers.
- However, k is small, so we can maintain each Champernowne word modulo k.
- When transitioning from the nth Champernowne word to the $(n+1)$th, we can multiply by 10^s and add $(n+1)$, where s is the number of digits in $n+1$. This should be maintained modulo k.
- Be careful about integer overflow, 64-bit integers suffice.
- Challenge: Can you solve this for small k but very large n?

Triangle Containment

Problem

- You are given a bunch of weighted points (x, y) in the plane. For each point, its value is defined as the sum of the weights of the other weighted points strictly inside the triangle defined by it, $(0, 0)$, and $(b, 0)$. Compute the value of every point.

Initial Observations

- n is too large to directly check, for each point, which points are strictly inside the induced triangle - it is possible to construct $\mathcal{O}(n^2)$ pairs where one point is inside the induced triangle by another point.
- If we sort the points by their directed angle θ_i around the origin, note that in order for point i to have point j inside its triangle, $\theta_j < \theta_i$.
- By similar logic, if we sort the points by their directed angle α_i around $(b, 0)$, we get a similar relation.

Triangle Containment

Solution

- Sort the point in reverse order by angle around $(b, 0)$.
- Looping over all points in this given order, we see that the points inside the current triangle must precede the current point. However, those points must also have θ smaller than the current point.
- We can maintain a segment tree keyed on index in the θ_i sort order. When we see point j, report the sum of all points seen so far with smaller θ, and then activate that point in the segment tree.
- Due to the large numbers, exact integer arithmetic must be used when sorting points by angle. This can be done by using cross products.

Color Tubes

Problem

- You have $n+1$ tubes each with the capacity to hold three balls. There are $3n$ balls distributed among the tubes, three balls each of n distinct colors. In a single move, you can take a ball from one tube and move it on top of all the other balls in a tube that has fewer than three balls in it. In $20n$ moves or fewer, get all tubes to be either completely empty or have all three balls of some color.

Solution

- There are many different approaches to get this to happen within $20n$ moves. We'll outline one approach that fills in the left n tubes. This solution will operate in multiple phases.

Initialization

- We start by emptying the rightmost tube, arbitrarily moving balls from there into tubes to the left that have space. This takes at most three moves.
- We proceed by making tube 1 be monochromatic, at which point future moves will not interact with it at all. We need to be able to perform this in fewer than 20 moves due to the overhead we incurred.

Making the Leftmost Tube Monochromatic

- Let the bottom ball in the leftmost tube have color c. We will move all balls with color c into this tube.
- If the tube is already monochromatic, we're done.
- If the topmost ball has color c and the middle one doesn't, we can reverse the two balls as follows:

Making the Leftmost Tube Monochromatic, continued

- Let the leftmost tube be l, the rightmost tube with balls be r, and the empty tube be e. Move a ball from r to e, the topmost ball with color c into e, the middle ball from l to r, the topmost ball with color c from e to l, and the last ball from e back to l. This takes five operations.
- Now, it remains to move balls from other tubes into the leftmost tube.
- If such a ball is not the bottom-most ball in its tube, we can remove the incorrect balls out of tube l into e, any balls above that ball into e, and then move that ball directly into l. Moving all balls back into e, this takes at most seven moves to fix one ball.
- If such a ball is the bottom-most ball in its tube, we can reverse the entire tube by moving all balls into tube e, at which point we can apply the above logic to move balls out of l until we can take the (now topmost ball) from e and move it into l. This takes at most eight moves.

Food Processor

Problem

- You have n different blades. Blade i can cut pieces of size at most m_i, cutting them in half in h_i seconds. Blades reduce the size at an exponential rate. Compute the minimum number of seconds needed to convert food that is originally size t to size s.

Solution

- For a given piece size, we want to use the blade with the minimal h_i rate. We can ignore blades where $m_i \leq s$ or $m_i > t$.
- We need to be able to solve the equation $t \cdot 0.5^{\frac{x}{h_i}} = s$ for x. Taking logarithms, we can show that $x = \frac{h_i \cdot \log\left(\frac{t}{s}\right)}{\log 2}$.
- We need to reevaluate the best blade for all m_i values in $[s, t]$. We can do this by maintaining the blades sorted by their m_i values. It is too slow to enumerate all eligible blades for each check.

Greedy Increasing Subsequences

Problem

- From a given sequence of integers, construct an increasing subsequence by taking the first integer from the sequence, and then taking the leftmost integer in the sequence greater than the most recently taken integer. Remove all these integers and repeat until the original sequence is empty. What sequences did you generate?

Solution

- Instead of generating the sequences in the order requested in the problem, we compute for each integer in the given input order which sequence it goes in.
- We can naively do this in $\mathcal{O}(n^2)$ by tracking the last integer in each sequence and looping over the sequences to find the earliest sequence where we can append to.
- Note that the last integers, when written in order, must be in nonincreasing order - if they were not, then the offending integer could have been added to an earlier sequence. We can therefore improve the performance of this algorithm to $\mathcal{O}(n \log n)$ by binary searching for the sequence to add it to instead of doing a linear scan.

Branch Manager

Problem

- In a rooted tree, people navigate through the tree by always traveling to the descendant with the lowest ID. n people start at the root and wish to get to specific destinations, traveling through the tree in order. Before each person starts traveling, you can permanently delete some edges from the tree. Compute the index of the first person who cannot make it home.

Initial Observations

- Use the Euler tour technique to represent the tree. Specifically, DFS through the tree in sorted order of children. Let s_v be the time when we first see vertex v in the DFS, and let e_v be the time when we return from vertex v in the DFS.
- We are therefore looking for the first vertex v where there exists a vertex u appearing before v in the destination order list where $e_v < s_u$.

Branch Manager

Solution

- If we compute the Euler tour of the tree, we can simply loop over the destination vertices in order, track the maximum s_v we have seen, and see when some e_v is less than the maximum e_v seen prior.
- Note that it is not strictly necessary to compute the Euler tour beforehand and then loop over the destination vertices in order. We can perform a preorder traversal of the tree. Prior to returning from the recursive call from a vertex v, we can visit any vertex that is in the call stack of the DFS, so we can loop over destination vertices until we see one we cannot visit.

I Could Have Won

Problem

- Alice and Bob are playing rock-paper-scissors - they each earn points with the first to earning k points winning a game, and points resetting to zero after. For what values of k does Alice win more games than Bob?

Solution

- Because the number of total points won by both Alice and Bob is at most $2 \cdot 10^3$, we can brute force all values of k up to the total number of points earned.
- We can directly simulate the result for a fixed value of k by maintaining the current count of points earned by both individuals as well as the number of games won by both individuals.

Sun and Moon

Problem

- The sun and the moon align for an eclipse occasionally. It was d_s years ago when the sun was last in the right place, and d_m years ago when the moon was last in the right place. The sun is in the right place once every y_s years, and the moon is in the right place once every y_m years. When will the next eclipse happen?

Solution

- We are guaranteed that an eclipse will happen in the next 5000 years.
- Therefore, we can check the years starting from one year in the future and check if the sun and moon will be in the right place - y is a valid year for an eclipse if $(y + d_m)$ is divisible by y_m and $(y + d_s)$ is divisible by y_s.
- There is a faster solution using the Chinese Remainder Theorem, but this was not required to solve the problem.

Everything is a Nail

Problem

- You are given a ternary array. You are to construct a ternary array where all 0's are contiguous, all 1's are contiguous, and all 2's are contiguous. Maximize the number of indices where your constructed array matches the given array.

Solution

- There are $\mathcal{O}(n^2)$ different ternary arrays you can construct, so checking all of them is too slow.
- However, if we construct our ternary array from left to right, the only information that matters is what integers have been used so far in our constructed ternary array and what the last added element is.
- Therefore, with dynamic programming, we can maintain the maximum number of integers we can match conditioned on having assigned the first i integers, the set of ternary integers we have used so far, and what the ith integer in our ternary array is.

Family Visits

Problem

- Given $n \leq 1\,000$ days, the amount of mess is increased by m_i each morning, and you can clean c_i amount of mess in the afternoon, determine the minimum number of afternoons you have to spend cleaning so that on d queried nights the mess is zero.
- Divide the days into segments in which the family visits on the last day. Each segment is then an independent subproblem of the same type. In each segment, the mess should be zero by the end of the last day.

Family Visits

Solution 1: Greedy

- On the last afternoon, if the mess is greater than zero, then you must clean on the last day; if the mess is already zero by then, you don't have to clean on the last day and can save an option of cleaning on the last day to remove mess created on previous days.
- Now consider a previous afternoon when the mess is greater than zero, you will have an option to clean on that day, along with all the cleaning options that you saved for the following days. Among those options, you should pick the cleaning with the largest c_i, until the mess becomes zero.

Family Visits

Solution 1: Greedy

- This yields a greedy solution working backwards: Initialize an empty cleaning option set S and total mess $t = 0$. For each day in reverse, add m_i to t and c_i to S. If $t > 0$, pick the largest values from S to reduce t to zero. The number of values picked corresponds to the number of afternoons spent cleaning.
- If we maintain S using a BBST or a heap, this greedy algorithm runs in $O(n \log n)$. The low constraints of the problem also allows you find the max value from S in linear time, so that $O(n^2)$ also passes.

Family Visits

Solution 2: DP

- Let $f(i,k)$ be the max amount of mess we can have starting on day i, such that we can clean k times in the following days and have no mess by the end of the last day. Assume the last day is day n.
- $f(i,k) < 0$ means it's impossible to clean all mess by day n. In terms of arithmetics we treat any negative value as negative infinity.
- We have:

$$f(i,k) = max \begin{cases} f(i+1,k) - m_i & \text{if } i < n \\ & \text{(don't clean on day } i\text{)} \\ f(i+1,k-1) + c_i - m_i & \text{if } i < n, k > 0 \\ & \text{(clean on day } i\text{)} \\ c_i - m_i & \text{if } i = n \text{ and } k > 0 \\ -m_i & \text{if } i = n \text{ and } k = 0 \end{cases}$$

Family Visits

Solution 2: DP

- Find the smallest k as our final answer such that $f(1, k) \geq 0$. This can be done by iterating k incrementally.
- There are $O(n^2)$ DP states in total and the transition takes constant time. Therefore the DP solution runs in $O(n^2)$ time and space.